47장. 에러 처리

47.1 에러 처리의 필요성

에러가 발생하지 않는 코드를 작성하는 것은 불가능. try... catch 문을 사용해 발생한 에러에 적절하게 대응하면 프로그램이 강제 종료되지 않고 계속해서 코드를 실행시킬 수 있다.

```
// 47-01
console.log('[start]');

try {
  foo();
} catch (error) {
  console.log('[에러 발생]', error);
  // [에러 발생] ReferenceError: foo is not defined
}

// 발생한 에러에 적절한 대응을 하면 프로그램이 강제 종료되지 않는다.
console.log('[End]');
```

예외적인 상황에 적절하게 대응하지 않으면 에러로 이어질 가능성이 크다.

```
// 47-03
// DOM에 button 요소가 존재하지 않으면 querySelector 메서드는 에러를 발생시키지 않고 n
ull을 반환한다.
const $button = document.querySelector('button'); null

$button.classList.add('disabled');
// TypeError: Cannot read property 'classList' of null
```

위 예제의 querySeletor 메서드는 인수로 전달한 문자열이 CSS 선택자 문법에 맞지 않는 경우 에러를 발생시킨다.

```
// 옵셔널 체이닝 연산자 `?.`을 사용해서 에러를 막음
const $button = document.querySeletor('button'); // null
$button?.classList.add('disabled');
```

47.2 try...catch...finally 문

에러 처리를 구현하는 방법

1. `querySelector`나 `Array#find` 메서드처럼 예외적인 상황이 발생하면 반환하는 값을 if 문이나 단축 평가 또는 옵셔널 체이닝 연산자를 통해 확인해서 처리하는 방법
2. 에러 처리 코드를 등록해 두고 에러가 발생하면 에러 처리 코드로 점프하도록 하는 방법이 있다.

```
try {
  // 실행할 코드(에러가 발생할 가능성이 있는 코드)
} catch (err) {
  // try 코드 블록에서 에러가 발생하면 이 코드 블록의 코드가 실행된다.
  // err 에는 try 코드 블록에서 발생한 Error 객체가 전달된다.
} finally {
  // 에러 발생과 상관없이 반드시 한 번 실행된다.
}
```

47.3 Error 객체

`Error 생성자 함수`에는 에러를 상세히 설명하는 에러 메시지를 인수로 전달할 수 있다.

```
const error = new Error('invalid');
```

Error 생성자 함수가 생성한 에러 객체는 message 프로퍼티와 stack 프로퍼티를 갖는다. message 프로퍼티의 값은 Error 생성자 함수에 인수로 전달한 에러 메시지이고, stack 프로퍼티의 값은 에러를 발생시킨 콜스택의 호출 정보를 나타내는 문자열이며 디버깅 목적으로 사용한다.

47.4 throw 문

에러 객체 생성과 에러 발생은 의미가 다르다.

에러를 발생시키려면 try 코드 블록에서 throw 문으로 에러 객체를 던져야 한다.

`throw 표현식;`

throw 문의 표현식은 어떤 값이라도 상관없지만 일반적으로 에러 객체를 지정.

에러를 던지면 catch문의 에러 변수가 생성되고 던져진 여러 객체가 할당된다. 그리고 catch 코드 블록이 실행되기 시작

47.5 에러의 전파

에러는 호출자 방향으로 전파된다. **즉, 콜 스택의 아래방향으로 전파된다.**

```
const foo = () => {
  throw Error('foo에서 발생한 에러'); // 4
}

const bar = () => {
  foo(); // 3
}
```

```
const baz = () => {
  bar(); //2
}

try {
  baz() // 1
} catch (err) {
  console.error(err);
}
```

1에서 baz함수를 호출하면 2에서 bar함수가 호출되고 3에서 foo함수가 호출되고 foo함수는 4에서 에러를 throw 한다. 이때 foo함수가 throw한 에러는 다음과 같이 호출자에게 전파되어 전역에서 캐치된다.

> 전역 실행 컨텍스트 <- baz 실행 컨텍스트 <- bar 실행 컨텍스트 <- foo 실행 컨텍스트

throw된 에러를 캐치하여 적절히 대응하면 프로그램을 강제 종료시키지 않고 코드의 실행 흐름을 복구할 수 있다. throw된 에러를 어디에서도 캐치하지 않으면 프로그램은 강제 종료된다.